Zoea

Research note 34

Towards a Self-Optimising Codebase

Edward McDaid & Sarah McDaid
7 Nov 2024

Optimisation of Zoea knowledge sources could be automated using evolutionary programming. This is facilitated by Zoea's blackboard architecture.

Evolutionary approaches to software generation have an innate appeal - largely due to their parallels with natural selection in biology. Perhaps mechanisms similar to those that created most forms of life could also be useful for the production of software. These approaches are also relatively straightforward, with no requirement for huge training sets, complex models or even much in the way of explicit domain knowledge. Behind the metaphors though, fundamentally they represent different forms of search. Indeed, such approaches have often been applied to problems with very large search spaces.

There are a range of different evolution-inspired approaches of which evolutionary programming is arguably the most flexible, since it imposes no constraints on the structure of the solution. Outlined conceptually, the principles of operation are simple and relatable. For any given problem we can have one or potentially many starting points. A population is comprised of many diverse individuals that come into and out of existence over time. Evolution occurs over a number of generations through changes called mutations. Each mutation is an incremental change derived from one or more individuals in the previous generation. Mutations that succeed are rewarded by continued existence and the opportunity to engage in reproduction.

So how could this type of approach be applied to Zoea? For one thing, Zoea also deals with a problem with an enormous search space. An obvious strategy might be to use evolutionary programming to directly support search for code solutions, in some way. A

more interesting proposition is to apply the approach to improve and extend the Zoea codebase itself.

Zoea employs a population of problem solving components called knowledge sources. Indeed, knowledge sources represent most of the Zoea codebase. Individually, each knowledge source encapsulates some aspect of programming language or software development knowledge. For example, there are many knowledge sources that deal with different kinds of conditional logic, looping constructs, type conversion and so on. Zoea knowledge sources are organised in a blackboard architecture - which basically means that they can all share information with one another, where it is mutually relevant.

Zoea knowledge sources have a very regular structure. Each one corresponds to a different kind of code fragment, that may or may not be present in a solution for any given problem. In functional terms, every knowledge source transforms a set of input test cases into a set of output test cases. In doing this, it converts the problem it is required to solve into a different problem, which is often smaller and/or simpler. The simpler problem is then made available for all of the knowledge sources to consider, and so on. If a solution to a simpler problem is found then the corresponding code fragment may be relevant in the context of its parent knowledge source. Finding a complete code solution in Zoea can be thought of as assembling the right hierarchy of knowledge sources. Currently, the code for these knowledge sources is fixed at any given point in time. However, it doesn't have to be.

It is not particularly difficult to create or modify Zoea knowledge sources. Indeed, all of the information required exists in the corresponding code fragment. The first step is to transform the code into a kind of template, where some of the code elements are replaced by slots. Next, we must define the data interfaces, both for the fragment as a whole and for each slot. The data flow model of the code fragment can be used to identify variable roles with respect to test case inputs and outputs. The template code represents its own operational semantics with respect to the transformation of test case inputs into outputs.

Knowledge source modification can be accomplished simply by changing of the corresponding code and regenerating. Creation and modification of knowledge sources from code fragments is both guided and constrained by Zoea heuristics, including

instruction subsets and digrams. The deletion of knowledge sources is trivial - other than for the analysis of any associated impact.

The existing set of Zoea knowledge sources were created manually. This is how explicit knowledge externalisation in Zoea is accomplished. While the existing knowledge sources work well as they are, it is also strongly suspected that they are not optimal in terms of their scope, range and composition. This is because a number of such optimisations have already been undertaken manually. For example, we can add or modify a knowledge source to allow specific types of problems to be solved more quickly. Such changes can in turn make other knowledge sources obsolete - increasing the amount of work required for all problems. The various changes to knowledge sources that are possible can be characterised as create, delete, split, merge, expand and narrow.

Unfortunately, it can be difficult to identify these optimisation opportunities manually and while the work involved is not difficult, it is impossible to estimate any benefits or predict the associated impact in advance. This makes the activity somewhat experimental in nature and labour intensive. Furthermore, the modification of knowledge sources also entails a significant amount of regression testing. As a result, automation of this optimisation process would be highly beneficial.

An evolutionary programming approach, using the existing knowledge sources as a starting point, has the potential to deliver this kind of optimisation automatically and iteratively. This would involve the introduction of a proportion of mutated knowledge sources within an otherwise relatively stable population. In this approach, mutations would only be created at the start of a new problem, so as to ensure that they are not at a disadvantage with respect to existing knowledge sources. Mutations would also need persist across a number of complete problem solving cycles in order to facilitate a proper evaluation.

Such mutated knowledge sources are not expected to be successful very often but occasionally they will be. This can include the cases where we might otherwise not have found a solution with given resources, where we find a solution using less resources or where we find a solution that is better in some way. We can defer the problem of regression testing by baselining the set of existing knowledge sources and versioning any changes. In this way, we can either regression test later or do so continuously - either in a

dedicated environment or a separate virtual blackboard. Any decision on whether to accept or reject a particular mutation does not need to be made immediately. Relaxing the strict imposition of synchronised generations is more like real life in any event.

In this scheme, the population of existing knowledge sources effectively competes with a percentage of additional mutants during the formulation of each solution. The number of mutants created simply depends on the amount of resources that are to be invested in knowledge source improvement. In theory, this approach will continue to improve the codebase indefinitely, although the benefits are expected to reduce over time. Measurement of the time and resources required to complete the Zoea regression test pack should quantify the level of improvement achieved and also indicate when investment is no longer generating a sufficient return.

In terms of finding code solutions for users' problems, it is interesting to note that this approach represents a completely separate additional form of search. On one hand we have the regular Zoea search for a code solution, using a given set of knowledge sources. We also have a different search activity - to find an improved set of knowledge sources - that happens over a longer clock cycle. The search space of the latter is the actual mechanism for the former. Effectively, this is a form of two-level search.

On first glance, it would be natural to assume that we are only making extra work for ourselves by adding an additional level of search in this way. However, the size of the search space for knowledge sources is a tiny fraction of the size of the search space for complete code solutions. This is because most code solutions employ more than one knowledge source and these can be combined in many different ways. Also, for any given problem there are some subsets of (all possible) knowledge sources that will allow a solution to be found more easily than with others. This two-level search helps to ensure we are not always stuck using the same - likely sub-optimal - subset of knowledge sources.

In any event, with this approach Zoea will continue to find solutions at least as well as it does today. It is also guaranteed that all solutions produced will function correctly with respect to the user specified test cases. For the cost of some fixed level of overhead, Zoea's performance will gradually improve, without further intervention. In this sense, the approach has some similarities to a Gödel machine. The actual overhead depends on the

chosen level of investment and consequently the proportion of mutations that are to be created.

Generating a completely new set of knowledge sources from scratch is an interesting theoretical possibility but likely to be impractical for various reasons. Rather, the existing set of knowledge sources act as a bootstrap so we begin with a system that already works. It would seem that - given a sufficient critical mass of knowledge - this kind of system is capable of finishing its own development by itself. This approach is made significantly easier by the blackboard architecture, which allows existing and mutated knowledge sources to co-exist and compete, and guarantees that working solutions will continue to be found.

Learn more at **zoea.co.uk**